Findrange

De Maio Dario, Mauro Nicola, Santangelo Angelo

January 2023

INDICE

1 Introduzione

A tutti è capitato di dover cambiare telefono almeno una volta nella vita.
Ciò che a prima vista potrebbe sembrare una semplice operazione è, nella realtà dei fatti, un'attività molto più complicata di quello che potrebbe sembrare.

Questo perché al fine di effettuare la scelta migliore bisogna considerare diversi fattori, primo fra tutti: il prezzo.
Ognuno di noi, infatti, mira al risparmio, ma allo stesso tempo necessita di un dispositivo che possiede determinate caratteristiche capaci di soddisfare le proprie esigenze. Tuttavia non sempre è facile riuscire a conciliare tali caratteristiche con una fascia di prezzo che rientri nelle proprie possibilità economiche.

A tal fine sarebbe utile avere un tool che date in input le caratteristiche di proprio interesse, sia in grado di fornire un'indicazione della fascia di prezzo che un tale device potrebbe avere. In tal modo si potrebbe decidere di effettuare una spesa consci del valore effettivo del prodotto, oppure "sacrificare" una caratteristica in favore del risparmio, nella speranza che così facendo si riesca ad individuare un dispositivo più accessibile al nostro portafoglio.

Da questi presupposti nasce l'idea di "Findrange".
Il nostro studio non si focalizzerà solo sull'implementazione di "Findrange", ma anche e soprattutto sull'analisi dei risultati forniti dalle varie tecniche di machine learning che impiegheremo per la creazione del progetto.
In altre parole, il nostro lavoro sarà utilizzare una serie di dati di test per allenare e valutare ciascun algoritmo, quindi confrontare i risultati ottenuti e infine determinare quale algoritmo gode delle prestazioni migliori.

2 Obiettivi

Il prezzo di un dispositivo può variare in base a diverse caratteristiche e proprietà dello stesso; l'identificazione di una più congeniale fascia di prezzo all'interno della quale l'acquirente si vuole limitare è un aspetto fondamentale.

Lo scopo del progetto "Findrange" è quello di realizzare diversi agenti intelligenti addestrati sulla base di un dataset e capaci di stabilire una fascia di prezzo ottimale, il che induce a pensare che occorra impiegare un algoritmo di classificazione.

Il criterio alla base è di partire da un insieme di caratteristiche del dispositivo scelte dall'utente ed individuare, con diversi modelli, la fascia di prezzo relativa al dispositivo avente quelle determinate caratteristiche richieste.

La fase successiva si articola in una valutazione dei risultati ottenuti dai diversi modelli al fine di stabilire quale si presti meglio all'esecuzione di questo compito.

3 PEAS

Perfomance	La misura di perfomance dell'agente è la sua capacita di categorizzare correttamente il range del dispositivo. Questa verrà valutata tenendo in considerazione le metriche di precisione, accuratezza e recall.
Enviroment	L'ambiente in cui opera l'agente è quello della telefonia mobile caratterizzato dalle proprietà di ogni dispositivo come memoria, qualità della fotocamera, ecc. L'ambiente è: • **Statico**, in quanto nel corso delle elaborazioni dell'agente, l'ambiente resta invariato; • **Episodico**, in quanto le azioni passate non influenzano le decisioni future dell'agente e la scelta dell'azione in ciascun episodio dipende dall'episodio stesso; • **Discreto**, in quanto il range del prezzo di un prodotto assume un numero limitato di valori: 4; • **Completamente osservabile**, in quanto si ha accesso a tutte le informazioni relative ai dispositvi in ogni momento; • **Non deterministico**, in quanto lo stato successivo dell'ambiente non è determinato nè dallo stato corrente nè dalle azioni dell'agente; • **A singolo agente**, in quanto l'unico agente che opera in questo ambiente è quello in oggetto.
Actuators	Gli attuatori dell'agente sono i modelli di machine learning addestrati.
Sensors	Il sensore dell'agente è rappresentato dalla tastiera (tramite essa l'agente riceve gli input) e dal dataset che contiene le informazioni relative alle caratteristiche.

4 Identificazione delle risorse necessarie: tool e linguaggi

I tool utilizzati

• KAGGLE: è una piattaforma online per l'analisi dei dati e il machine learning. Offre una vasta gamma di dati, modelli di machine learning e strumenti di sviluppo per aiutare gli utenti a creare soluzioni di data science e machine learning. E' utile per la raccolta dei dataset, sia privati che pubblici, su diverse tematiche che gli utenti possono utilizzare per esplorare, analizzare e sviluppare modelli di machine learning. Nel nostro caso è stato usato proprio tale scopo.

• VISUAL STUDIO CODE: è un editor di codice sorgente che offre un'interfaccia utente intuitiva e una serie di funzionalità avanzate per la scrittura e il debugging del codice, come il supporto per diversi linguaggi di programmazione e la gestione dei controlli del codice sorgente. E' particolarmente adatto per lo sviluppo di progetti di piccole e medie dimensioni. Tuttavia, offre anche una serie di estensioni e plugin che possono essere utilizzati per estenderne le funzionalità e adattarlo a diverse esigenze di sviluppo. Tale ambiente è stato usato, nel nostro caso, per la stesura del codice sorgente, ovvero per l'implementazione del modello di machine learning.

• JUPYTER NOTEBOOK: è un'applicazione web open source che consente di creare e condividere documenti che contengono codice, equazioni, visualizzazioni e testo. I documenti creati utilizzando Jupyter Notebook vengono chiamati "notebook". I notebook sono molto utilizzati nel campo della scienza dei dati e dell'analisi dei dati, poiché permettono di eseguire il codice in modo interattivo e di visualizzare i risultati direttamente nel documento. Inoltre, i notebook consentono di documentare il processo di lavoro in modo dettagliato, rendendo facile la riproduzione dei risultati e la condivisione del lavoro con altre persone. Nel nostro caso lo abbiamo usato proprio per riprodurre risultati di grafici in tempo reale, così da analizzare da subito eventuali problematiche legate ai dati, ma anche per documentare le azioni effettuate, le quali hanno consentito una maggior chiarezza durante la condivisione del lavoro con gli altri membri del team di progetto.

Il linguaggio utilizzato

• PYTHON: è un linguaggio di programmazione ad alto livello e a scopo generico, cioè è un linguaggio che può essere utilizzato per sviluppare qualsiasi tipo di software, dai sistemi operativi ai giochi, dai siti web alle applicazioni per il mobile. Python è noto per la sua sintassi semplice e leggibile, che rende il codice facile da scrivere e da mantenere. Inoltre, Python include una vasta libreria standard che offre funzionalità di alto livello per molti compiti comuni di sviluppo, come la gestione dei file, la connessione ai database, creazione di modelli di machine learning, ecc. Nel nostro caso, è stato usato per la stesura del codice sorgente, ovvero per l'implementazione dell'algoritmo di machine learning.

5 Librerie

Per il completo funzionamento del nostro progetto, si è deciso di utilizzare le seguenti librerie offerte da Python:

- **pandas** : fornisce strutture e strumenti per l'analisi di dati in linguaggio Python. La sua principale struttura di dati è il DataFrame, che è simile a un foglio di calcolo con righe e colonne.

- **seaborn**: fornisce un'interfaccia intuitiva per creare grafici statistici e visualizzazioni dei dati, ed è particolarmente utile per l'analisi dei dati multivariati.

- **numpy**: fornisce funzioni matematiche avanzate per lavorare con array multidimensionali, come algebra lineare e generazione di numeri casuali.

- **matplotlib.pyplot**: una sottolibreria di matplotlib che fornisce una interfaccia per creare in Python grafici e visualizzazioni di diverse tipologie come linee, barre, istogrammi, scatter plots, grafici a torta e molto altro ancora.

- **matplotlib_venn**: una libreria per la creazione di diagrammi di Venn utile per la visualizzazione di dati relazionali tra insiemi e per la rappresentazione di sovrapposizioni e differenze tra gruppi di elementi. La funzione venn3 consente di creare un diagramma di Venn a 3 insiemi.

- **joblib**: una libreria per la memorizzazione su disco e il caricamento veloce di oggetti Python.
  La funzione dump consente di salvare un oggetto Python su disco in un formato compresso e può essere utilizzata per salvare modelli di machine learning che si desidera mantenere per un utilizzo futuro.
  La funzione load consente di caricare un oggetto serializzato (salvato in precedenza) da un file su disco e ripristinarlo in memoria.

- **sklearn**: fornisce un'ampia gamma di algoritmi di apprendimento automatico, funzioni di preprocessing, di valutazione, e di selezione modello. Nello specifico sono stati utilizzati i seguenti moduli:

  - **prepocessing**: fornisce alcuni trasformatori e classificatori che vengono utilizzati per preparare i dati per l'addestramento e la valutazione di modelli di apprendimento automatico.
    Ad esempio la classe MinMaxScaler che è un trasformatore per normalizzare i dati in un intervallo compreso tra 0 e 1.
  - **model_selection**: fornisce funzioni per la selezione dei modelli di apprendimento automatico.
    Ad esempio la funzione train_test_split che fornisce la suddivisione dei dati in training set e test set.

- **naive_bayes**: fornisce un insieme di classificatori Naive Bayes. I classificatori Naive Bayes sono algoritmi di apprendimento automatico che utilizzano il teorema di Bayes per la classificazione e sono particolarmente utili in quanto possono essere addestrati utilizzando un gran numero di proprietà (caratteristiche) dei dati.
  Ad esempio la classe GaussianNB che implementa un classificatore Naive Bayes Gaussiano che assume che tutte le proprietà sono distribuite normalmente (gaussiane) e che sono indipendenti tra di loro.
- **tree**: fornisce un insieme di classi e funzioni per l'utilizzo di alberi decisionali come modelli di classificazione e regressione. Ad esempio la classe DecisionTreeClassifier che implementa un modello di classificazione basato su alberi decisionali. Un albero decisionale è un grafico che rappresenta tutte le possibili decisioni in un problema di classificazione. Ogni nodo interno dell'albero rappresenta una proprietà o una caratteristica dei dati, mentre ogni foglia rappresenta una classe. Il modello è addestrato sui dati di addestramento utilizzando un algoritmo di apprendimento automatico e poi può essere utilizzato per classificare nuovi dati.
- **metrics**: fornisce una serie di funzioni per la valutazione dei modelli di apprendimento automatico, come ad esmpio: precision_score, accuracy_score, recall_score che calcolano rispettivamente precisione, accuratezza e recall.
- **neural_network**: fornisce una serie di algoritmi di apprendimento automatico basati su reti neurali. In particolare, fornisce classi per la costruzione e l'allenamento di reti neurali per il problema di classificazione e di regressione. Una di queste è MLPClassifier: una classe per la costruzione e l'allenamento di una rete neurale multi-livello (Multi-layer Perceptron, MLP) per il problema di classificazione.
- **ensemble**: fornisce una serie di algoritmi di apprendimento automatico che utilizzano la combinazione di più modelli per prendere una decisione, in modo da sfruttare la diversità e la robustezza dei modelli individuali. VotingClassifier è una classe per la costruzione e l'allenamento di un classificatore di voto, che combina più classificatori in un unico modello. Il classificatore di voto utilizza il voto dei classificatori individuali per prendere una decisione sulla classe di un oggetto di esempio.

6 Identificazione del dataset

L'identificazione del giusto dataset rappresenta un punto cardine nella progettazione di un modello capace di fare previsioni, in quanto i dati devono essere conformi agli obiettivi di business da soddisfare. Risulta, quindi, necessario assicurarsi che siano coerenti con il problema reale.

I possibili approcci per identificare un dataset sono due:

1. Creazione di un dataset, utilizzando competenze e conoscenze personali.
2. Utilizzare un dataset pubblico, già verificato e adattarlo all'esigenze del team di lavoro.

La prima soluzione presenta alcune problematiche legate alla scarsità dei dati a disposizione, alla potenziale infondatezza o imprecisione di essi, poiché basati su esperienze personali e soggettive, ed infine, i costi legati al tempo di creazione.

Per l'identificazione del dataset da utilizzare si è optato per la seconda soluzione, attendibile e verificata, ma ugualmente ardua, data la presenza sul web dei numerosi dataset poco veritieri o strutturati in modo non coerente e di difficile elaborazione.

Grazie a Kaggle, uno dei siti più famosi per la distribuzione di insiemi di dati, è stato possibile reperire un dataset coerente rispetto al problema in esame, ricco di dati e affidabile.

Nelle fasi successive, attraverso vari tool, sarà svolta un'analisi del dataset e una pulizia dei dati al fine di migliorarne la qualità così da garantire un apprendimento migliore ai modelli.

7 Documentazione dei dati

I dati ottenuti dal dataset mostrano le seguenti caratteristiche:

1. **battery_power**: indica l'energia della batteria del dispositivo misurata in mAh;
2. **blue**: indica se il dispositivo ha (valore 1) o no (valore 0) il bluetooth;
3. **clock_speed**: indica la velocità di clock del dispositivo;
4. **dual_sim**: indica se il dispositivo è (valore 1) o no (valore 0) dual sim;
5. **fc**: indica i megapixel della camera frontale del dispositivo;
6. **four_g**: indica se il dispositivo ha (valore 1) o no (valore 0) il 4G;
7. **int_memory**: indica la capacità della memoria interna del dispositivo espressa in gigabyte;
8. **m_dep**: indica lo spessore del dispositivo espresso in cm;
9. **mobile_wt**: indica il peso del dispositivo;
10. **n_cores**: indica il numero di core del processore del dispositivo;
11. **pc**: indica il numero di pixel della fotocamera principale del dispositivo;
12. **px_height**: indica la risoluzione dei pixel in altezza del dispositivo;
13. **px_width**: indica la risoluzione dei pixel in larghezza del dispositivo;
14. **ram**: indica la RAM del dispositivo espressa in megabyte;
15. **sc_h**: indica l'altezza del dispositivo espressa in cm;
16. **sc_w**: indica la larghezza del dispositivo espressa in cm;
17. **talk_time**: indica la durata massima (in ore) della batteria mentre si è in chiamata col dispositivo;
18. **three_g**: indica se il dispositivo ha (valore 1) o no (valore 0) il 3G;
19. **touch_screen**: indica se il dispositivo è (valore 1) o no (valore 0) touch-screen;
20. **wifi**: indica se il dispositivo ha (valore 1) o no (valore 0) il wi-fi;
21. **price_range**: indica la fascia di prezzo del dispositivo;

La nostra variabile indipendente è price_range che può assumere valori compresi tra 0 e 3.
Nello specifico, la fascia di prezzo è così distribuita:
Range 0 per valori compresi tra 200 e 350 euro
Range 1 per valori compresi tra 351 e 500 euro
Range 2 per valori compresi tra 501 e 650 euro
Range 3 per valori compresi tra 651 e 800 euro

8 Data preparation

8.1 Data cleaning

Non sono presenti dati mancanti nel dataset, di conseguenza non abbiamo effettuato la fase di Data Imputation, ovvero la fase dove si stimano i valori dei dati mancanti. Inoltre non sono presenti neanche stringhe di testo, di conseguenza non c'era bisogno di effettuare una normalizzazione del testo o altre tecniche riguardanti il campo del Natural Language Processing.

8.2 Feature Selection

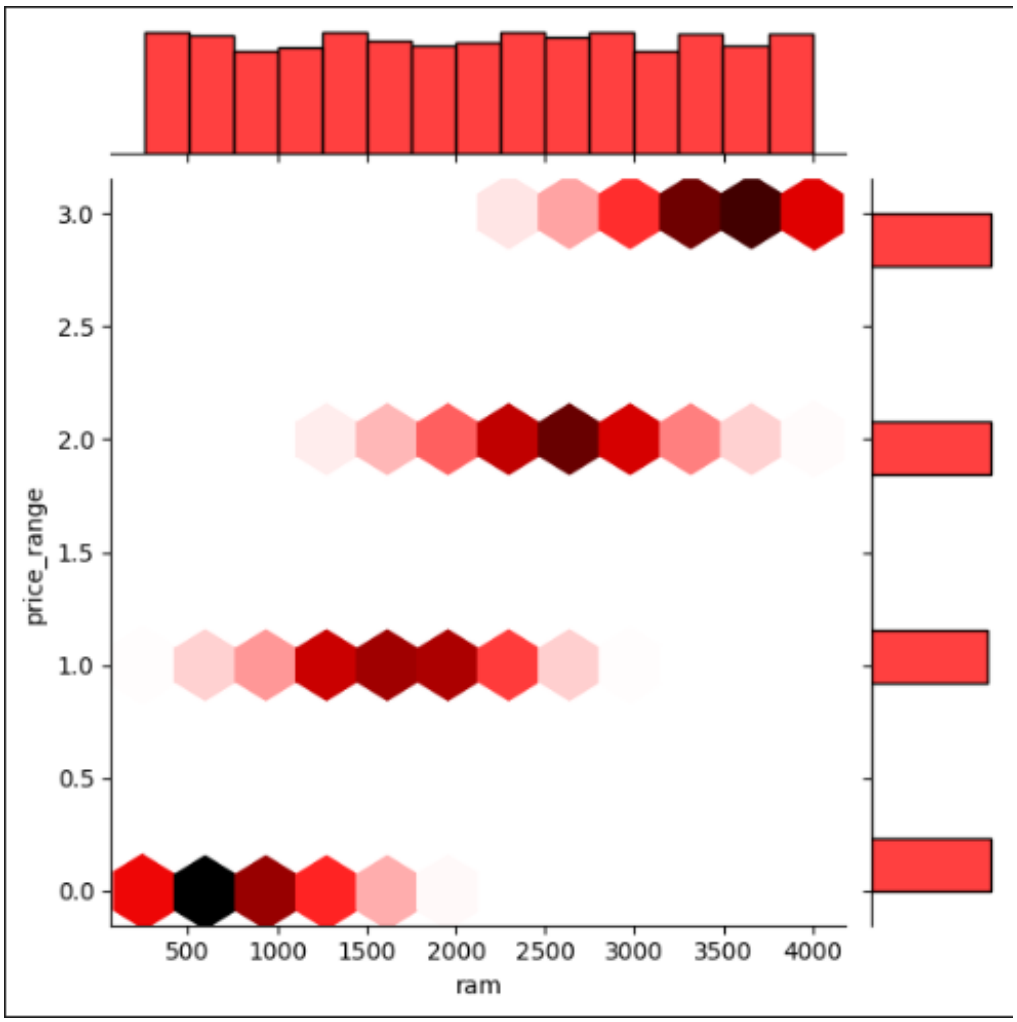
Questa è una delle fasi più importanti e fondamentali al fine di avere un buon machine learner, ovvero quello della selezione delle caratteristiche più correlate al problema in esame. Tale fase comprende la Feature Engineering, nella quale il progettista utilizza la propria conoscenza del dominio applicativo del problema per determinare le feature più rilevanti che possano caratterizzare gli aspetti principali del problema in esame e, quindi, influenzare la potenza predittiva del machine learner.
Per fare ciò, prima di tutto, abbiamo fatto uso della libreria "pandas", la quale offre il metodo **read_csv("dataset.csv")** per leggere un dataset: nel nostro caso si tratta di un file CSV (file di testo contenente una rappresentazione dati in forma tabulare), denominato "dataset.csv". Tale metodo restituisce proprio l'intera rappresentazione tabulare dei dati. Dopodichè abbiamo usato il metodo **describe()**, il quale restituisce un oggetto DataFrame contenente le statistiche descrittive calcolate. Ad esempio, se applicato a una serie di numeri, calcolerà la media, la deviazione standard, il minimo, il massimo, il quartile 1 (25%), il quartile 2 (50%) il quartile 3 (75%) e il numero degli elementi. Concatenato a **describe()**, abbiamo usato il metodo **transpose()**, il quale ci permette di scambiare righe e colonne di una matrice e di avere una migliore visualizzazione. Prima di fare ciò, abbiamo eliminato l'attributo **id** dal DataSet, in quanto non serviva ai fini del progetto.
Di seguito mostrato il funzionamento.

```
dataFrame = pd.read_csv("dataset.csv").drop(columns=["id"])

dataFrame.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
battery_power	3000.0	1241.849000	437.063804	500.0	863.75	1232.0	1619.00	1999.0
blue	3000.0	0.502000	0.500079	0.0	0.00	1.0	1.00	1.0
clock_speed	3000.0	1.528467	0.820358	0.5	0.70	1.5	2.30	3.0
dual_sim	3000.0	0.512000	0.499939	0.0	0.00	1.0	1.00	1.0
fc	3000.0	4.404000	4.383742	0.0	1.00	3.0	7.00	19.0
four_g	3000.0	0.510000	0.499983	0.0	0.00	1.0	1.00	1.0
int_memory	3000.0	32.581667	18.152810	2.0	16.00	33.0	48.00	64.0
m_dep	3000.0	0.507000	0.285969	0.1	0.20	0.5	0.80	1.0
mobile_wt	3000.0	140.003000	35.213809	80.0	109.00	140.0	170.00	200.0
n_cores	3000.0	4.456333	2.289361	1.0	2.00	4.0	6.00	8.0
pc	3000.0	9.962333	6.073923	0.0	5.00	10.0	15.00	20.0
px_height	3000.0	639.112333	440.202998	0.0	277.75	564.0	932.50	1960.0
px_width	3000.0	1247.601667	434.666168	500.0	865.00	1248.0	1634.00	1998.0
ram	3000.0	2129.141333	1085.694231	256.0	1212.75	2147.5	3065.25	3998.0
sc_h	3000.0	12.202667	4.251151	5.0	9.00	12.0	16.00	19.0
sc_w	3000.0	5.616667	4.322494	0.0	2.00	5.0	9.00	18.0
talk_time	3000.0	11.035667	5.474400	2.0	6.00	11.0	16.00	20.0
three_g	3000.0	0.759667	0.427357	0.0	1.00	1.0	1.00	1.0
touch_screen	3000.0	0.502000	0.500079	0.0	0.00	1.0	1.00	1.0
wifi	3000.0	0.507000	0.500034	0.0	0.00	1.0	1.00	1.0
price_range	3000.0	1.508333	1.122059	0.0	0.00	2.0	3.00	3.0

Dopodichè, abbiamo proseguito con il controllo di quanto, vari attributi presenti nel dataset, influiscano sull'esito della variabile dipendente "price_range".
Iniziamo con il calcolare il grado di influenza dell'attributo "ram". Per fare ciò abbiamo fatto uso di una funzione, offerta dalla libreria "seaborn", che si chiama **jointPlot()**, la quale crea un diagramma di giunzione, che mostra la relazione tra due variabili quantitative su uno sfondo di densità di probabilità: nel nostro caso la prima variabile rappresenta l'attributo del quale vogliamo testarne l'influenza, mentre la seconda variabile rappresenta la variabile dipendente, ovvero "price_range".
Di seguito, mostrato il funzionamento.

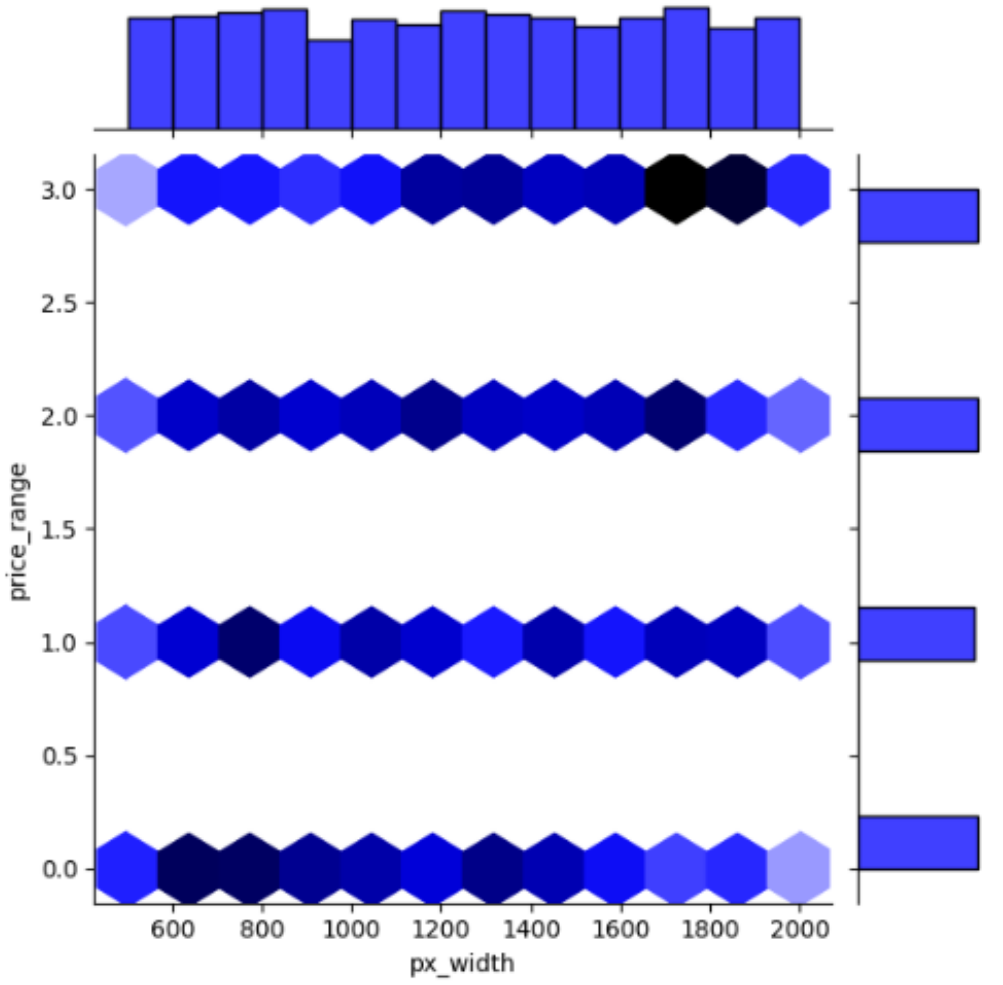
```
sns.jointplot(x='ram', y='price_range', data=dataFrame, color='red', kind='hex')
```



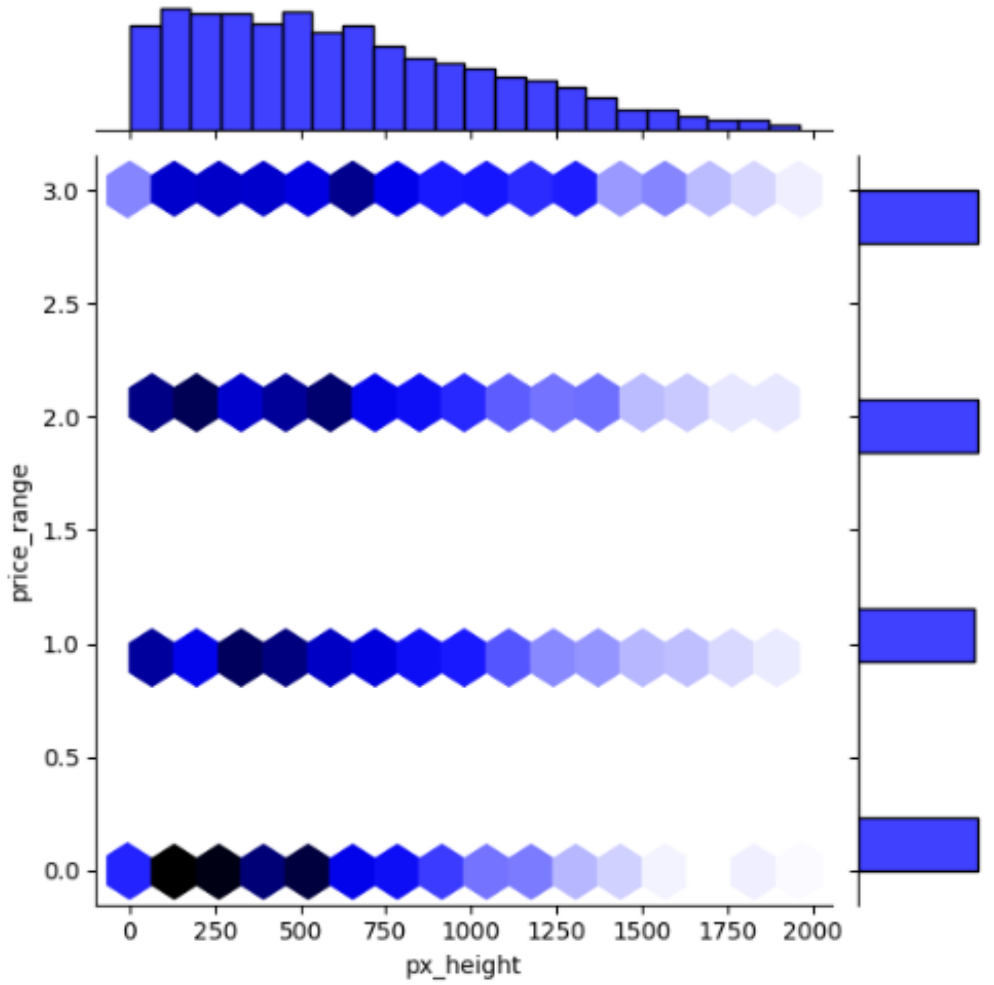
In questo caso la RAM influisce tanto in quanto, si può vedere dal grafico, che al crescere della RAM (asse delle ascisse), il range da predire (asse delle ordinate) tende ad aumentare.
Di conseguenza non elimineremo tale attributo dal Dataset.

Calcoliamo il grado di influenza di 'px_height' e 'px_width', i quali rappresentano informazioni circa la risoluzione dei pixel in altezza e larghezza del telefono.

```
sns.jointplot(x='px_width', y='price_range', data=dataFrame, color='blue', kind='hex')
```




```
sns.jointplot(x='px_height', y='price_range', data=dataFrame, color='blue', kind='hex')
```

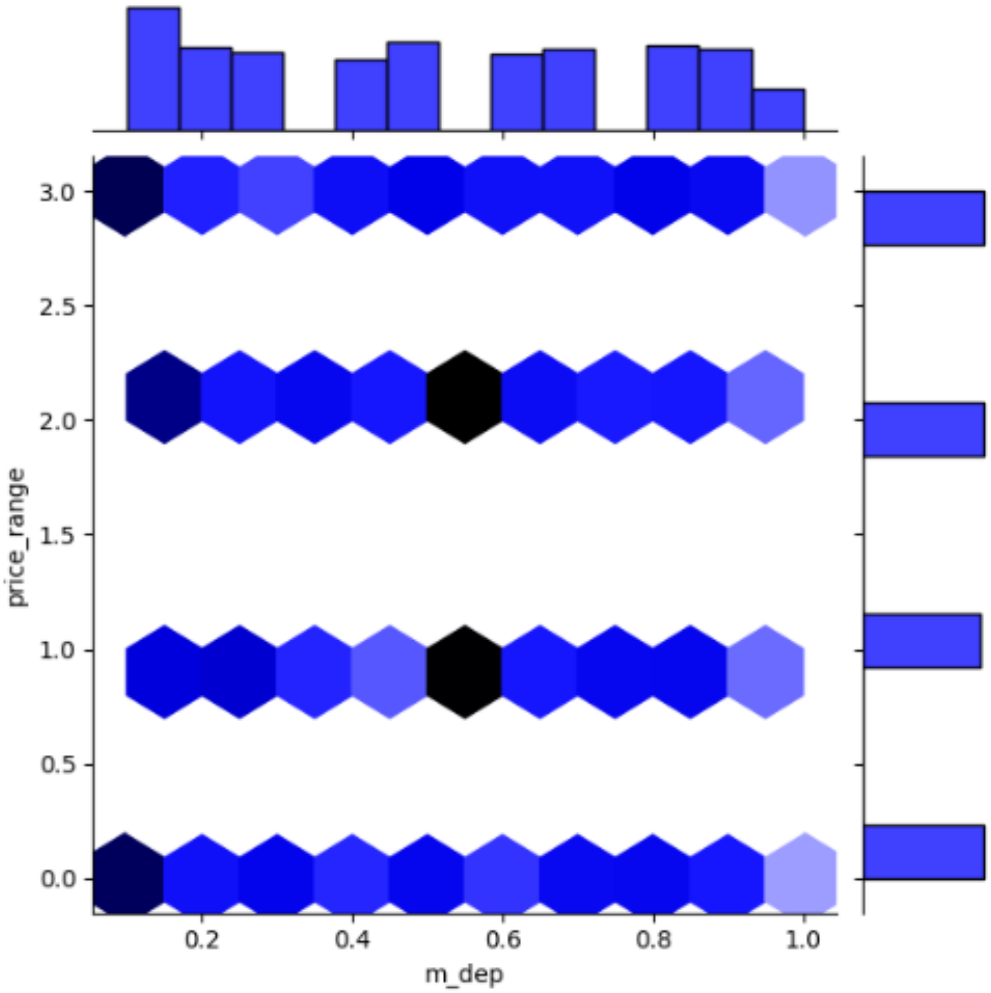


La risoluzione in pixel del dispositivo non influisce affatto.
Di conseguenza rimuoveremo 'pixel_width' e 'pixel_height' dal Dataset.

```
dataFrame.drop(columns=["px_width", "px_height"], inplace=True)
```

Calcoliamo, ora, il grado di influenza dell'attributo 'm_dep', il quale rappresenta la profondità del telefono cellulare.

```
sns.jointplot(x='m_dep', y='price_range', data=dataFrame, color='blue', kind='hex')
```

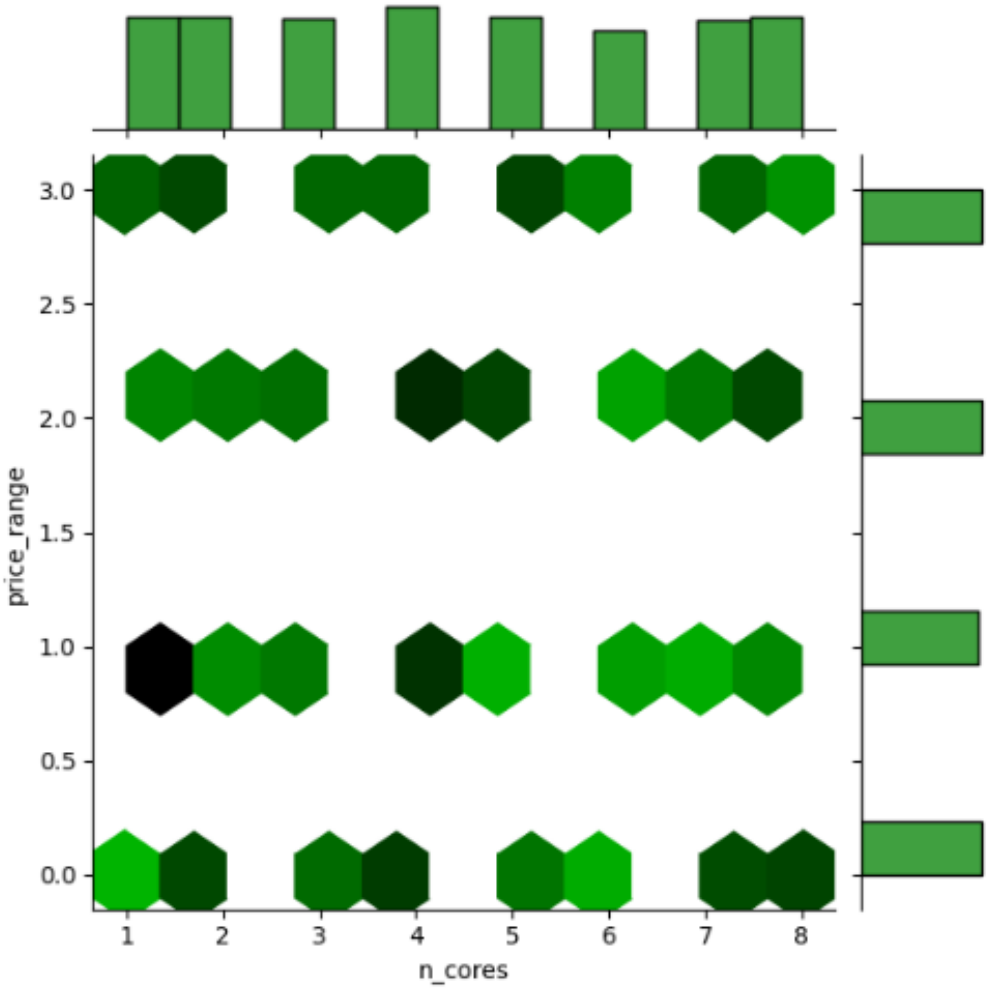


La profondità non influisce affatto.
Di conseguenza, provvederemo a rimuovere l'attributo 'm_dep' dal Dataset.

```
dataFrame.drop(columns=["m_dep"], inplace=True)
```

Continuiamo con il calcolare il grado di influenza dell'attributo 'n_cores', il quale rappresenta il numero di processori del telefono cellulare.

```
sns.jointplot(x='n_cores', y='price_range', data=dataFrame, color='green', kind='hex')
```

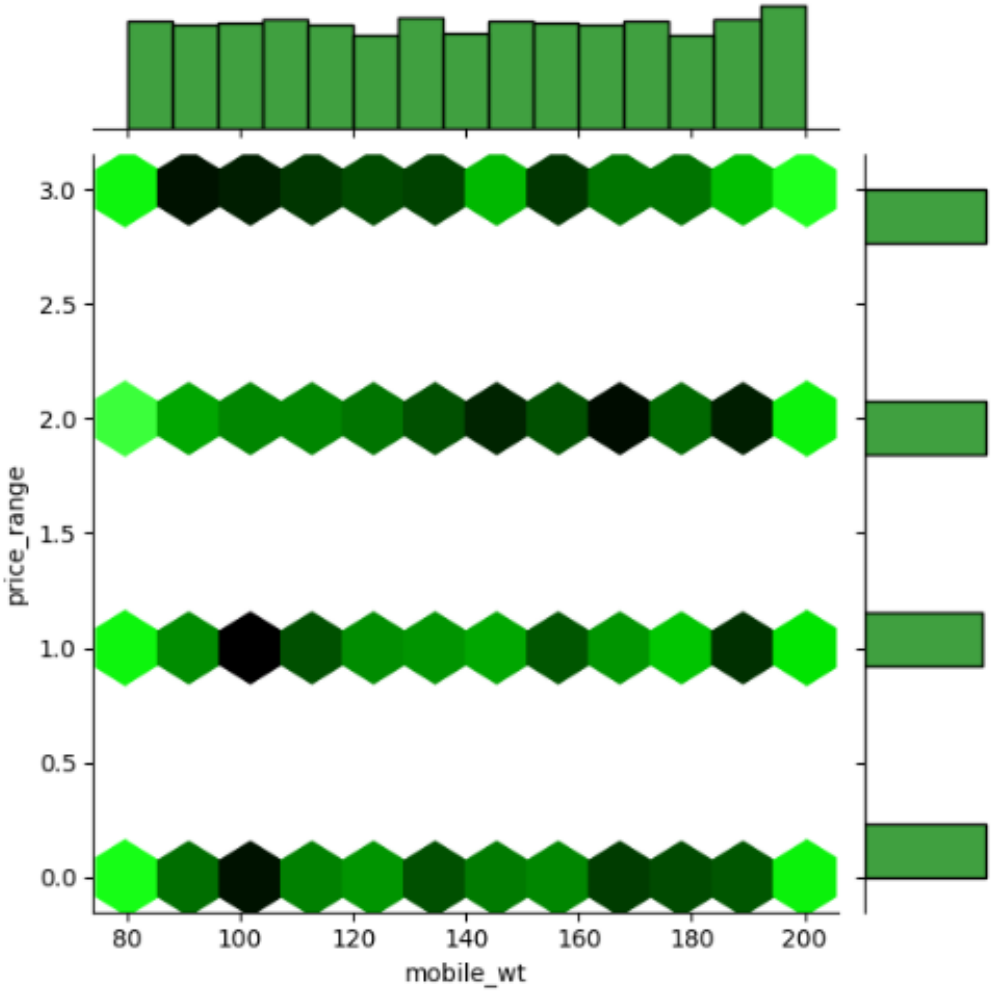


Il numero di processori non influisce affatto.
Di conseguenza, provvederemo all'eliminazione di 'n_cores' dal Dataset.

```
dataFrame.drop(columns=["m_dep"], inplace=True)
```

Calcoliamo il grado di influenza dell'attributo 'mobile_wt', il quale rappresenta la larghezza del telefono cellulare.

```
sns.jointplot(x='mobile_wt', y='price_range', data=dataFrame, color='green', kind='hex')
```

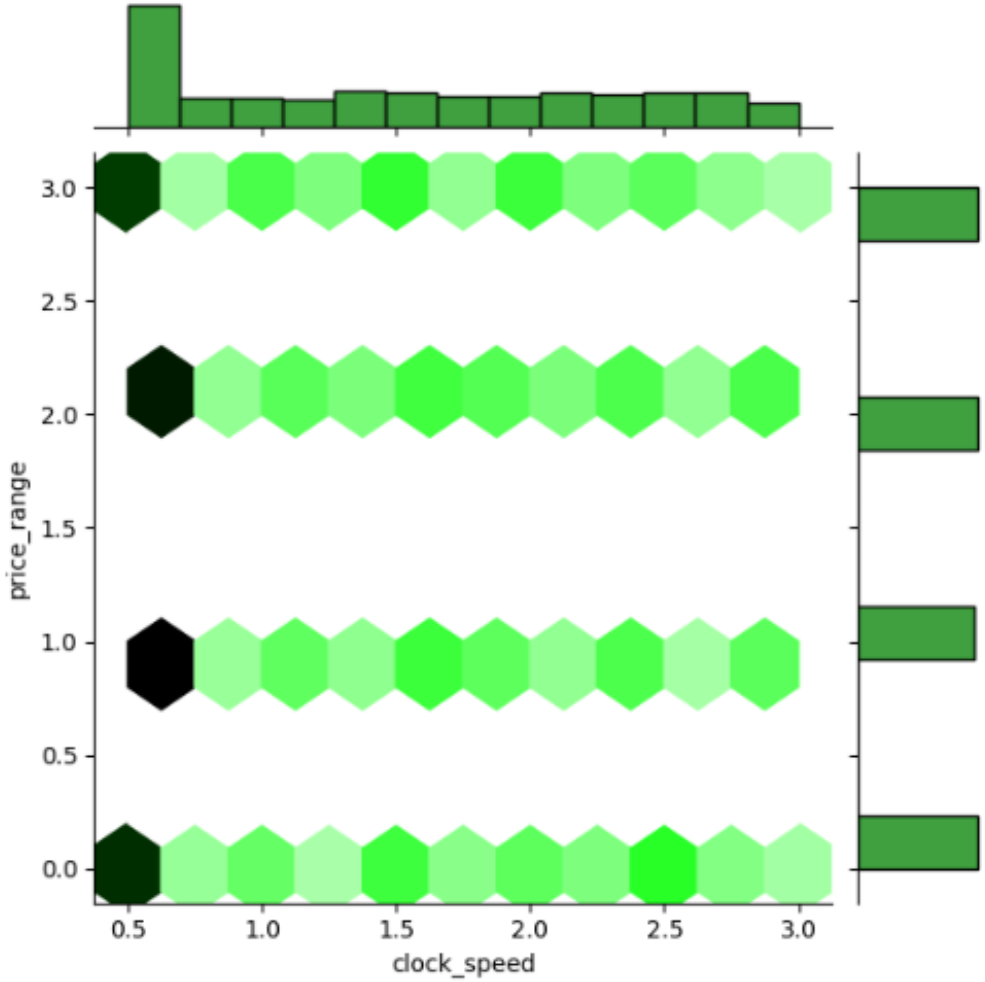


La larghezza del dispositivo non influisce.
Di conseguenza, provvederemo all'eliminazione di 'mobile_wt' dal Dataset.

```
dataFrame.drop(columns=["mobile_wt"], inplace=True)
```

Calcoliamo il grado di influenza dell'attributo 'clock_speed', il quale rappresenta la velocità con cui il microprocessore esegue le istruzioni.

```
sns.jointplot(x='clock_speed', y='price_range', data=dataFrame, color='green', kind='hex')
```

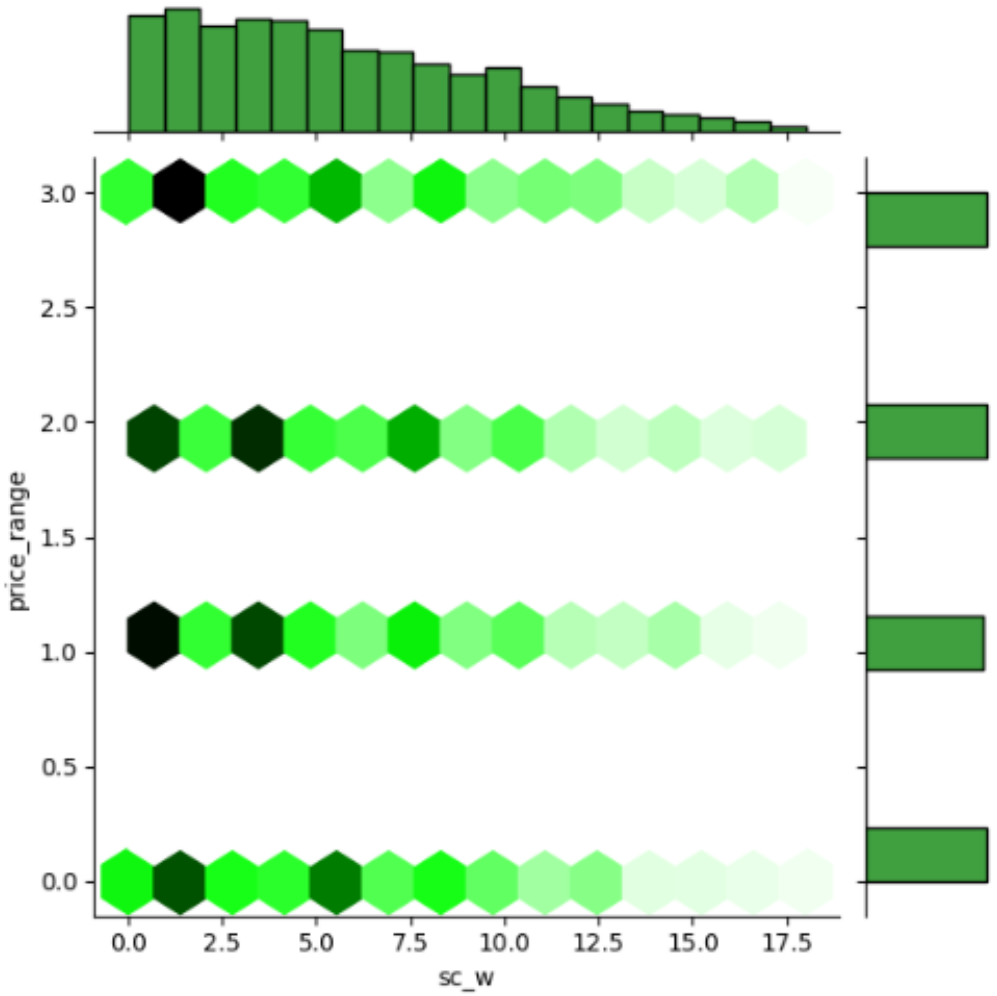


La velocità del microprocessore non influisce.
Di conseguenza, provvederemo all'eliminazione di 'clock_speed' dal Dataset.

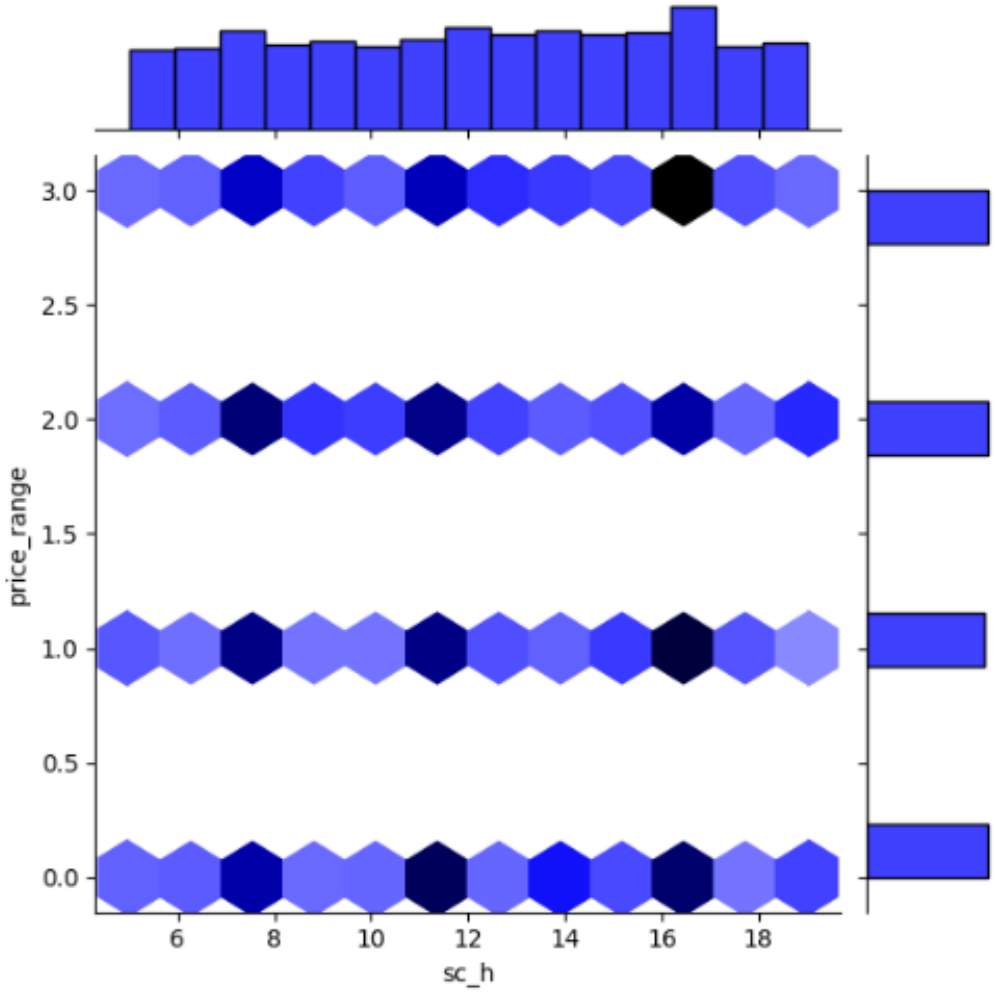
```
dataFrame.drop(columns=["clock_speed"], inplace=True)
```

Calcoliamo il grado di influenza degli attributi 'sc_w' e 'sc_h', i quali rappresentano la larghezza e l'altezza dello schermo del telefono (in cm).

```
sns.jointplot(x='sc_w', y='price_range', data=dataFrame, color='green', kind='hex')
```




```
sns.jointplot(x='sc_h', y='price_range', data=dataFrame, color='blue', kind='hex')
```



La larghezza e altezza dello schermo non influiscono tanto.
Di conseguenza provvederemo all'eliminazione di 'sc_w' e 'sc_h' dal Dataset.

```
dataFrame.drop(columns=["sc_w", "sc_h"], inplace=True)
```

Quando i cellulari vengono pubblicizzati, il numero dei megapixel indicato, di solito, è in realtà la somma dei megapixel della fotocamera frontale e di quella posteriore.
Nel nostro Dataset, l'attributo "fc" rappresenta i megapixel della frontal camera e "pc" rappresenta i megapixel della "primary camera".
Andremo quindi a definire un nuovo attributo "camera", il quale rappresenterà la somma di "pc" e "fc" per ogni riga del Dataset.

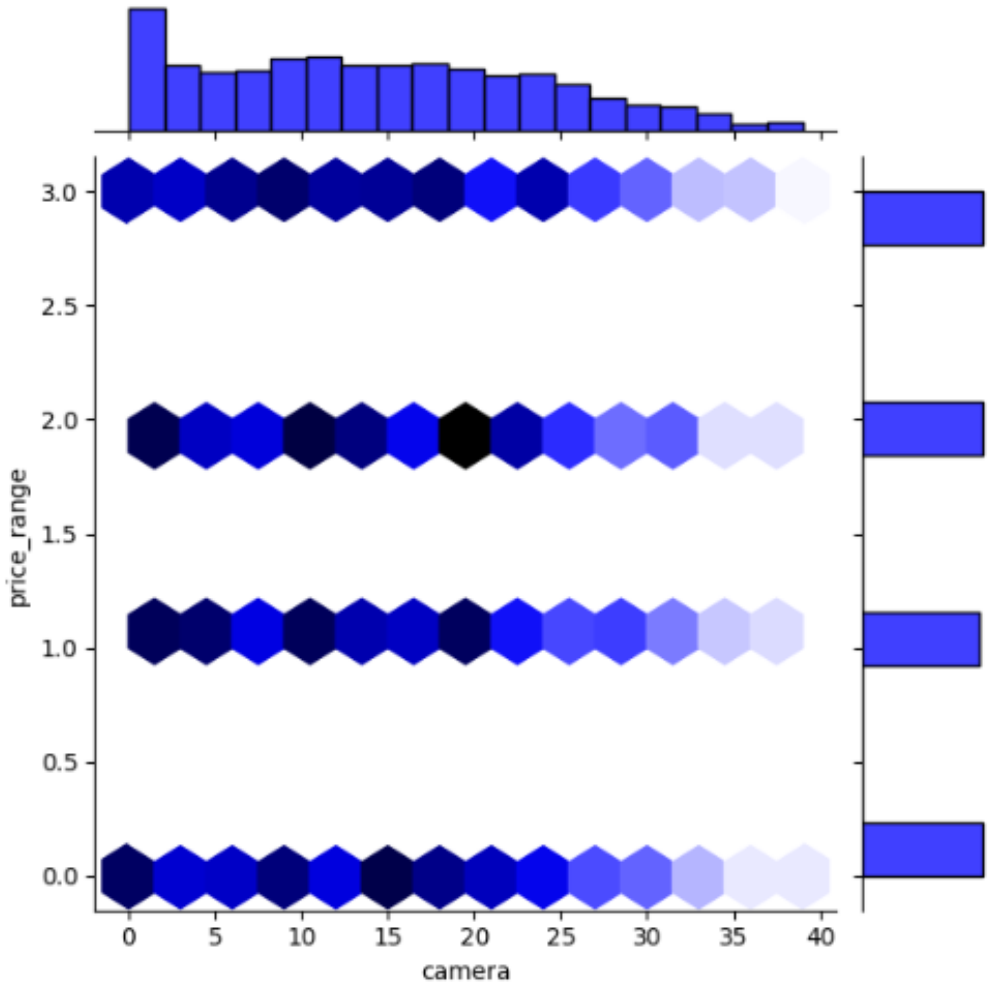
```
dataFrame["camera"] = dataFrame["fc"] + dataFrame["pc"]
dataFrame.drop(columns=["fc", "pc"], inplace=True)
dataFrame
```

Vediamo a video il nuovo Dataset contenente la nuova colonna "camera" che accorpa "pc" e "fc".

	battery_power	blue	dual_sim	four_g	int_memory	ram	talk_time	three_g	touch_screen	wifi	price_range	camera
0	842	0	0	0	7	2549	19	0	0	1	1	3
1	1021	1	1	1	53	2631	7	1	1	0	2	6
2	563	1	1	1	41	2603	9	1	1	0	2	8
3	615	1	0	0	10	2769	11	1	0	0	2	9
4	1821	1	0	1	44	1411	15	1	1	0	1	27
...	...	...	...	...	...	...	...	...	...	...	...	...
2995	1700	1	0	1	54	2121	15	1	1	0	2	17
2996	609	0	1	0	13	1933	19	0	1	1	1	2
2997	1185	0	0	1	8	1223	14	1	0	0	1	13
2998	1533	1	1	0	50	2509	6	0	1	0	2	12
2999	1270	1	0	1	35	2828	3	1	0	1	2	23

Calcoliamo, quindi, il grado di influenza del nuovo attributo 'camera', il quale rappresenta il numero dei megapixel totali della fotocamera anteriore e posteriore.

```
sns.jointplot(x='camera', y='price_range', data=dataFrame, color='blue', kind='hex')
```



Nonostante la poca influenza dell'attributo 'camera', terremo comunque in considerazione tale attributo.

Stampiamo ora il Dataset dopo alcune modifiche e cancellazioni delle variabili indipendenti.

	count	mean	std	min	25%	50%	75%	max
battery_power	3000.0	1241.849000	437.063804	500.0	863.75	1232.0	1619.00	1999.0
blue	3000.0	0.502000	0.500079	0.0	0.00	1.0	1.00	1.0
dual_sim	3000.0	0.512000	0.499939	0.0	0.00	1.0	1.00	1.0
four_g	3000.0	0.510000	0.499983	0.0	0.00	1.0	1.00	1.0
int_memory	3000.0	32.581667	18.152810	2.0	16.00	33.0	48.00	64.0
ram	3000.0	2129.141333	1085.694231	256.0	1212.75	2147.5	3065.25	3998.0
talk_time	3000.0	11.035667	5.474400	2.0	6.00	11.0	16.00	20.0
three_g	3000.0	0.759667	0.427357	0.0	1.00	1.0	1.00	1.0
touch_screen	3000.0	0.502000	0.500079	0.0	0.00	1.0	1.00	1.0
wifi	3000.0	0.507000	0.500034	0.0	0.00	1.0	1.00	1.0
price_range	3000.0	1.508333	1.122059	0.0	0.00	2.0	3.00	3.0
camera	3000.0	14.366333	9.523709	0.0	6.00	14.0	21.00	39.0

8.3 Feature Scaling

In questa fase abbiamo analizzato accuratamente il Dataset, al fine di identificare gli attributi aventi range di valori molto ampio e diversi tra loro.
Questo è uno dei passaggi più importanti e fondamentali di questa fase perché se l'insieme dei valori per una determinata caratteristica è molto diverso rispetto ad un altro, l'algoritmo potrebbe sottostimare/sovrastimare l'importanza di una caratteristica, il che porterebbe ad un conseguente confondimento del machine learner.
Osservando la stampa del Dataset, nella fase precedente, possiamo notare che, ad esempio, l'attributo "battery_power" possiede un range di valori che va dai '501.0' a '1998.0', il che si riflette in una distribuzione standard molto elevata. Questo lo si può vedere anche, ad esempio, per l'attributo "ram". Tuttavia, questo non vale per attributi che assumono solo uno tra i valori 0 e 1, i quali rappresentano, rispettivamente, 'No' e 'Si'. L'obiettivo, quindi, è quello di normalizzare tutti gli attributi del dataset, in modo tale che coloro i quali assumono solo uno tra i valori 0 e 1, rimarranno invariati, mentre tutti gli altri verranno normalizzati nell'intervallo [0, 1].
La Feature Selection ci mette a disposizione varie tecniche per normalizzare l'insieme dei valori, ovvero restringere il range dei numeri.
Uno dei metodi più conosciuti è il **MinMax Normalization**, usato anche da noi nel nostro caso, per normalizzare tutti gli attributi del dataset. Abbiamo innanzitutto droppato la colonna "price_range" dal DataFrame, creandone uno nuovo, contenente, quindi, tutte le variabili indipendenti, tranne quella dipendente e, quest'ultima, la salviamo in una variabile 'y', che la rappresenterà per le

successive operazioni di implementazione degli algoritmi.
Di seguito mostrato il funzionamento.

```
dataFrame2 = dataFrame.drop(columns=["price_range"])
y = dataFrame["price_range"]
```

Dopodichè abbiamo usato il metodo **MinMaxScaler()**, il quale è un metodo di scaling delle caratteristiche che viene utilizzato per trasformare un insieme di dati in modo che i valori vengano scalati in un intervallo specifico, nel nostro caso [0-1]. Nel codice, creiamo un oggetto della classe MinMaxScaler che viene utilizzato per trasformare un set di dati.
Successivamente, tramite il metodo **fit()** calcoliamo i valori minimi e massimi dei dati da utilizzare per la scalatura.
Infine il metodo **transform()** viene utilizzato per applicare la trasformazione di scalatura ai dati che sono stati adattati con il metodo fit(). Prende i dati di input e li scala in base ai valori minimi e massimi calcolati durante il passo di adattamento. L'output del metodo di trasformazione è un nuovo array con i valori scalati nell'intervallo definito [0-1].
Di seguito, mostrato il funzionamento.

```
normalizer = MinMaxScaler()
transformer = normalizer.fit(dataFrame2)
normalizedDataFrame = transformer.transform(dataFrame2)
```

Stampiamo il DataSet avente tutti i campi normalizzati nell'intervallo [0-1]

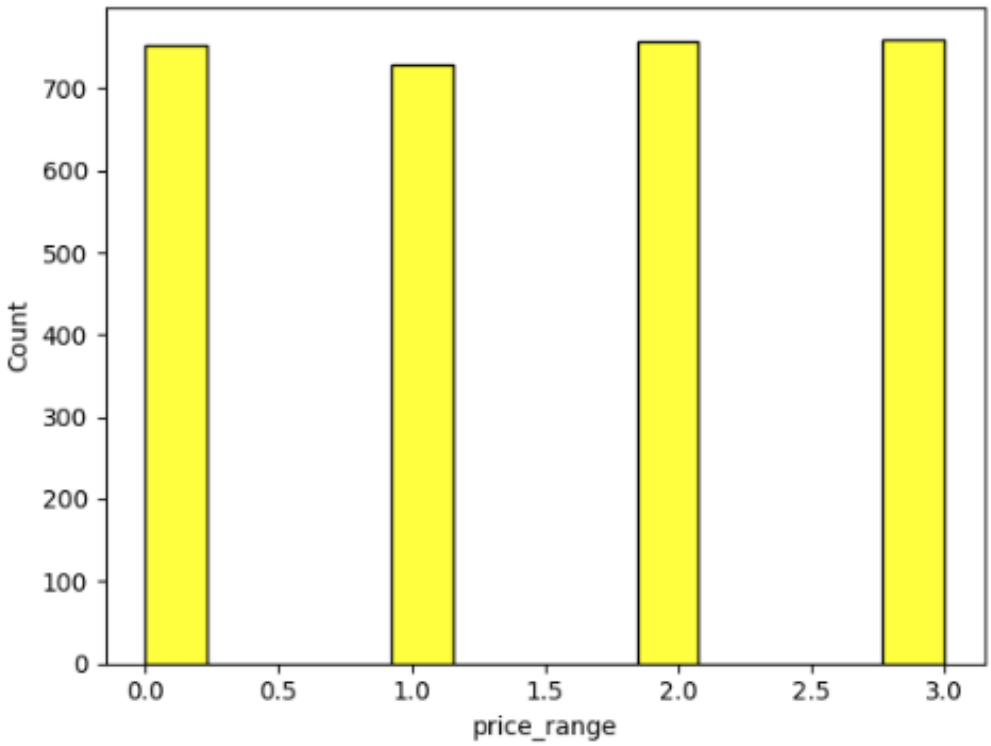
```
pd.DataFrame(normalizedDataFrame, columns=dataFrame.drop(columns=["price_range"]).columns).describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
battery_power	3000.0	0.494896	0.291570	0.0	0.242662	0.488326	0.746498	1.0
blue	3000.0	0.502000	0.500079	0.0	0.000000	1.000000	1.000000	1.0
dual_sim	3000.0	0.512000	0.499939	0.0	0.000000	1.000000	1.000000	1.0
four_g	3000.0	0.510000	0.499983	0.0	0.000000	1.000000	1.000000	1.0
int_memory	3000.0	0.493253	0.292787	0.0	0.225806	0.500000	0.741935	1.0
ram	3000.0	0.500572	0.290137	0.0	0.255679	0.505478	0.750735	1.0
talk_time	3000.0	0.501981	0.304133	0.0	0.222222	0.500000	0.777778	1.0
three_g	3000.0	0.759667	0.427357	0.0	1.000000	1.000000	1.000000	1.0
touch_screen	3000.0	0.502000	0.500079	0.0	0.000000	1.000000	1.000000	1.0
wifi	3000.0	0.507000	0.500034	0.0	0.000000	1.000000	1.000000	1.0
camera	3000.0	0.368368	0.244198	0.0	0.153846	0.358974	0.538462	1.0

8.4 Data Balancing

In questa fase l'obiettivo è quello di applicare tecniche per convertire un dataset sbilanciato in un dataset bilanciato. La motivazione legata a questa fase è il fatto che la maggior parte dei modelli di machine learning funzionano bene solo quando il numero di istanze di una certa classe è simile a quello di un'altra classe. Se non considerassimo il problema dello sbilanciamento dei dati, definiremmo un modello capace di caratterizzare bene solo gli esempi della classe più popolosa che, nella maggior parte dei casi, è quella che non ci interessa.
Nel nostro caso, per verificare ciò, abbiamo fatto uso di una funzione offerta dalla libreria "seaborn", denominata **histplot**, la quale ci dà la possibilità di tracciare istogrammi univariati o bivariati per mostrare le distribuzioni dei set di dati. Di seguito, mostrato l'output sull'istogramma in base alla variabile dipendente "price_range".

```
sns.histplot(dataFrame["price_range"], legend=True, color="yellow")
```



Possiamo osservare che è ben bilanciato: le righe del dataset sono 3000 e i valori totali che assume la variabile dipendente sono quattro (0, 1, 2, 3), quindi avere per ogni valore, un numero di istanze che si aggira intorno ai 740, implica avere una buona distribuzione dei valori per ogni tipo di range.
Di conseguenza non abbiamo dovuto effettuare tecniche di Data Balancing, come oversampling o undersampling, per bilanciare il nostro dataset.

9 Metriche selezionate

Per valutare la bontà delle predizioni effettuate dagli algoritmi che andremo ad utilizzare, impiegheremo tre metriche:

1. precision: indica la percentuale di predizioni effettuate dal modello che sono corrette rispetto al totale delle predizioni
2. accuracy: indica la proporzione di esempi positivi (o veri positivi) su tutti gli esempi che il modello ha classificato come positivi
3. recall: è utile per valutare la capacità del modello di evitare falsi negativi, ovvero di non classificare come negativo un esempio che in realtà è positivo.

10 Realizzazione training e test set

Ottenuti i dati normalizzati, la prima operazione in termini di addestramento è stata la divisione del nostro dataset "normalizedDataFrame" in training set e test set. Si è deciso di utilizzare il 67% dei dati per il training set e il restante 33% per il test set. Tale divisione verrà utilizzata da tutti gli algoritmi di machine learning che verranno implementati.

```
X_train, X_test, Y_train, Y_test = train_test_split(normalizedDataFrame, y, test_size=0.33, random_state=0)
```

11 Naive Bayes

Uno degli approcci che si è scelto per risolvere il problema è l'applicazione del classificatore Naive Bayes.
Tale scelta è giustificata dal fatto che la natura del problema suggerisce l'applicazione di un classificatore di tipo probabilistico. Di fatti, è naturale pensare che fattori quali la fotocamera o la memoria ram possano incidere sulla probabilità che il dispositivo rientri in una particolare fascia di prezzo piuttosto che in un'altra.
L'algoritmo considera le caratteristiche della nuova istanza da classificare e calcola la probabilità che queste facciano parte di una classe tramite l'applicazione del teorema di Bayes.
L'algoritmo è chiamato naive (ingenuo) poiché assume che le caratteristiche non siano correlate l'una all'altra. Di conseguenza, l'algoritmo non andrà a valutare in fase di classificazione la potenziale utilità data dalla combinazione di più caratteristiche
Quindi si è costruito un classificatore Naive Bayes Gaussiano e lo si è addestrato con i valori del training set.
Infine lo si utilizza per predire i valori di "X_test".
Tali previsioni vengono salvate in "y_pred_nb".

```
gnb = GaussianNB()

y_pred_nb = gnb.fit(X_train, Y_train).predict(X_test)
```

A questo punto, verifichiamo quanti valori predetti in "y_pred_nb" non coincidono con i valori effettivamente presenti nel test set "Y_test".

```
print("Numero di punti etichettati erroneamente su un totale di %d punti : %d"% (X_test.shape[0], (Y_test != y_pred_nb).sum()))
```

```
Numero di punti etichettati erroneamente su un totale di 990 punti : 224
```

11.1 Metriche Naive Bayes

Precision

```
precision_score(Y_test, y_pred_nb, average='macro')
```

0.767792900551985

Accuracy

```
accuracy_score(Y_test, y_pred_nb)
```

0.7737373737373737

Recall

```
recall_score(Y_test, y_pred_nb, average='micro')
```

0.7737373737373737

12 Decision Tree

Dati i risultati non proprio ottimali del classificatore Naive Bayes, si è deciso di adoperare un altro tipo di classificatore che potesse essere adeguato al problema. Il secondo algoritmo al quale si è pensato, è l'albero decisionale.

Gli alberi decisionali sono un tipo di modello di machine learning che utilizza una struttura ad albero per fare previsioni o prendere decisioni basate su una serie di caratteristiche o attributi.
In particolare, un albero di decisione è formato da nodi di decisione e nodi foglia. I nodi di decisione rappresentano le decisioni da prendere basate sull'analisi di una o più caratteristiche, mentre i nodi foglia rappresentano le previsioni o le decisioni finali.
Per costruire un albero di decisione, il modello analizza il train set e individua le caratteristiche che hanno il maggiore impatto sulla variabile dipendente. Ad ogni nodo di decisione viene quindi assegnata una caratteristica e il modello crea ramificazioni per ogni valore possibile di questa caratteristica. Il processo continua finché non si arriva a un nodo foglia, ovvero a una previsione o a una decisione finale.

Similmente a quanto effettuato in precedenza, si è costruito un classificatore "DecisonTreeClassifier" e lo si è addestrato attraverso la funzione "fit()", come mostrato di seguito.

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, Y_train)
```

Successivamente, si procede all'invocazione del metodo "predict()", per effettuare le previsioni sui dati di test.

```
y_pred_dtc = dtc.predict(X_test)
```

A questo punto, verifichiamo quanti valori predetti in "y_pred_dtc" non coincidono con i valori effettivamente presenti nel test set "Y_test".

```
print("Numero di punti etichettati erroneamente su un totale di %d punti : %d"% (X_test.shape[0], (Y_test != y_pred_dtc).sum()))
```

```
Numero di punti etichettati erroneamente su un totale di 990 punti : 237
```

12.1 Metriche Decision Tree

Precision

```
precision_score(Y_test, y_pred_dtc, average='macro')
```

0.7554393282005276

Accuracy

```
accuracy_score(Y_test, y_pred_dtc)
```

0.7595959595959596

Recall

```
recall_score(Y_test, y_pred_dtc, average='micro')
```

0.7595959595959596

13 Multi-layer Perceptron

Al fine di migliorare le capacità predittive di "Findrange", si è deciso di impiegare un ulteriore modulo di machine learning, stavolta leggermente più complesso: una rete neurale.
I risultati ottenuti fino a questo momento erano accettabili, ma sia che si trattasse di Naive Bayes che Decisional Tree, il numero di errori superava sempre 200, ossia il 20,2% del trainig set.
Per minimizzare questo valore è stato scelto il classificatore Multi-layer Perceptron.
Un MLP è una tipologia di Rete Neurale Artificiale che consiste in più strati di neuroni artificiali interconnessi tra loro. Il primo strato è chiamato strato di input e rappresenta l'inserimento dei dati, gli strati intermedi sono chiamati strati nascosti e l'ultimo strato è chiamato strato di output.
Al fine di comprendere al meglio il funzionamento del modulo, occorre fare riferimento alle linee di codice qui riportate:

```
mlp = MLPClassifier(alpha=0.25, hidden_layer_sizes=(128, 4), random_state=1, max_iter=24000)
mlp.fit(X_train, Y_train)
```

```
y_pred_mlp = mlp.predict(X_test)
```

Il codice crea un oggetto di una classe Multi Layer Perceptron (MLP) utilizzando la classe MLPClassifier di scikit-learn.

Il parametro "alpha" specifica il coefficiente di regolarizzazione L2 utilizzato nell'algoritmo di addestramento. Il coefficiente di regolarizzazione cerca di evitare overfitting penalizzando i pesi elevati nel modello. Il coefficiente di regolarizzazione L2 introduce una penalità sui pesi del modello, facendo in modo che i pesi siano più piccoli. Ciò rende il modello meno complesso e più in grado di generalizzare. La penalità è data dalla somma dei quadrati dei pesi del modello moltiplicata per un fattore di regolarizzazione, detto lambda.

Il parametro "hidden_layer_sizes" specifica la dimensione degli strati nascosti nella rete neurale MLP, in questo caso (128, 4) indica che ci saranno due strati nascosti con 128 e 4 neuroni rispettivamente.

Il parametro "random_state" imposta un seme per la generazione casuale dei pesi iniziali nella rete neurale, in modo che i risultati possono essere riprodotti.

Il parametro "max_iter "specifica il numero massimo di iterazioni per convergere.

In pratica, ad ogni iterazione il modello cerca di "perfezionarsi" ricalcolando i valori erroneamente predetti tendendo in considerazione il coefficiente di regolarizzazione. Quando "alpha" è basso, i pesi impiegheranno più tempo per aggiornarsi e per questo occorre aumentare il numero di iterazioni.

Tenendo in considerazione quanto detto, gli specifici valori da noi riportati sono stati individuati grazie a sperimentazioni empiriche.

Infine, addestriamo il modello similmente a quanto fatto in precedenza.
A questo punto, verifichiamo quanti valori predetti in "y_pred_mlp" non coincidono con i valori effettivamente presenti nel test set "Y_test".

```
print("Numero di punti etichettati erroneamente su un totale di %d punti : %d"% (X_test.shape[0], (Y_test != y_pred_mlp).sum()))
```

```
Numero di punti etichettati erroneamente su un totale di 990 punti : 180
```

13.1 Metriche Multi-layer Perceptron

Precision

```
precision_score(Y_test, y_pred_mlp, average='macro')
```

0.8151241195690219

Accuracy

```
accuracy_score(Y_test, y_pred_mlp)
```

0.8181818181818182

Recall

```
recall_score(Y_test, y_pred_mlp, average='micro')
```

0.8181818181818182

14 Ensemble learning

L'ensemble learning è una tecnica di machine learning che consiste nel combinare più modelli di machine learning per ottenere risultati migliori rispetto a quelli ottenuti utilizzando un singolo modello. L'utilizzo di più modelli consente anche di ridurre l'overfitting, aumentando le capacità del modello di generalizzare sui dati sconosciuti. Ci sono diverse tecniche di ensemble learning, quella selezionata in questo progetto è una classificazione a votazione. Per utilizzare un classificatore a voto in "sklearn", è possibile utilizzare la classe VotingClassifier del modulo ensemble. Questa classe accetta come input una lista di classificatori e una strategia di voto. Nell'esempio che segue, viene mostrato un classificatore basato su votazione, che combina i tre modelli analizzati precedentemente: il primo basato sul Decision Tree, il secondo su Naive Bayes, mentre il terzo è il Multilayer Perceptron.

```
votingClassifier = VotingClassifier(estimators=[('Tree',dtc), ('Bayes',gnb), ('MLP',mlp)], voting='hard')
```

Le possibili strategie di voto sono due: hard e soft. La prima strategia fa sì che ogni classificatore dell'insieme fornisce una previsione binaria e la previsione finale del classificatore a voto viene ottenuta facendo la maggioranza dei voti. La strategia "soft", invece, prevede che ogni classificatore dell'insieme fornisce una probabilità per ogni classe possibile e la previsione finale viene ottenuta facendo la media o la somma delle probabilità. Dopo aver fatto ciò, nella fase successiva si addestrerà il modello attraverso il metodo "fit()" e si salveranno i risultati della previsione all'interno della variabile "y_pred".

```
votingClassifier.fit(X_train, Y_train)
y_pred = votingClassifier.predict(X_test)
```

A questo punto, verifichiamo quanti valori predetti in "y_pred" non coincidono con i valori effettivamente presenti nel test set "Y_test".

```
print("Numero di punti etichettati erroneamente su un totale di %d punti : %d"% (X_test.shape[0], (Y_test != y_pred).sum()))
```

```
Numero di punti etichettati erroneamente su un totale di 990 punti : 198
```

14.1 Metriche ensemble

Precision

```
precision_score(Y_test, y_pred, average='macro')
```

0.7955955543520848

Accuracy

```
accuracy_score(Y_test, y_pred)
```

0.798989898989899

Recall

```
recall_score(Y_test, y_pred, average='micro')
```

0.798989898989899

15 Conclusioni

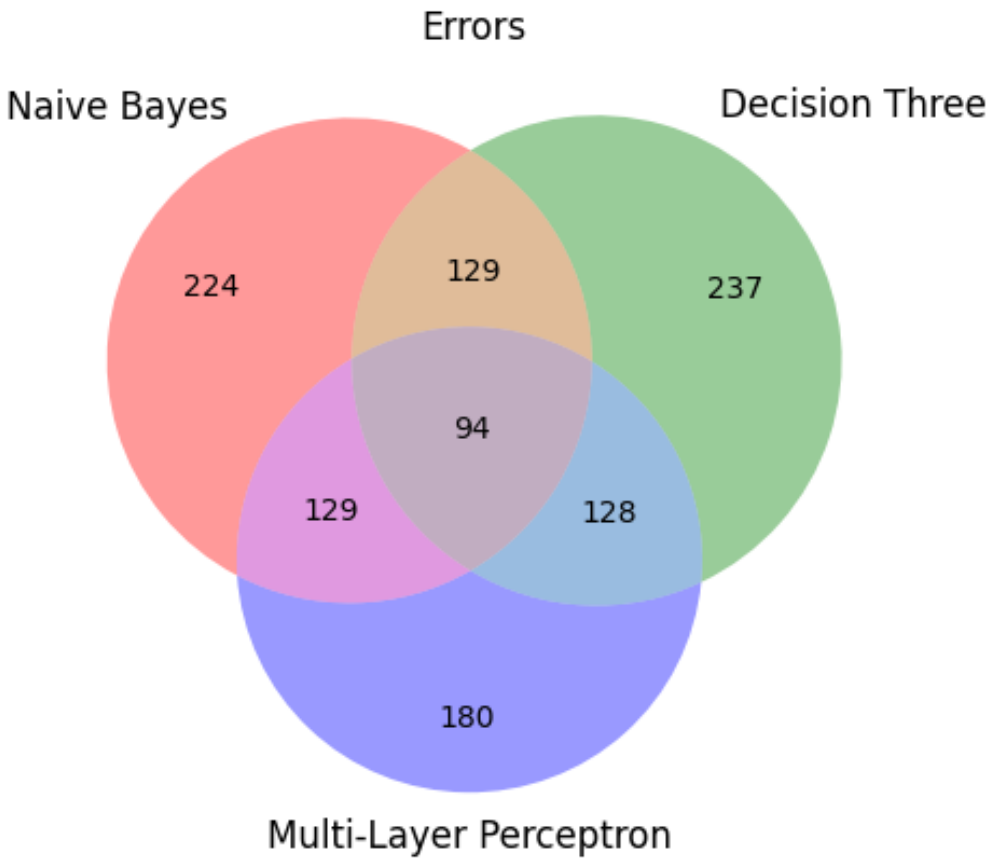
Terminato l'addestramento e dopo aver analizzato i risultati ottenuti, il modello basato sull'Ensemble learning non si presta ad essere il modello migliore, andando contro l'ipotesi iniziale del gruppo.
Durante le fasi alte della progettazione, sulla base di considerazioni empiriche, si ipotizzava che il modello di ensemble learning potesse essere quello **ideale**, in quanto permetterebbe di unire i punti di forza dei vari modelli che utilizza.
Per spiegare il motivo per il quale esso non si presta ad essere il migliore, sono state utilizzate funzioni e grafici, così da mappare gli errori in comune tra i vari modelli.
Come riportato in seguito, sono stati dichiarati tre vettori di errori, uno per ogni modello, contenenti gli errori commessi da quest'ultimi rispetto al test set. Gli errori vengono mappati attraverso il valore "1".

```
nb_errors = np.where(y_pred_nb == Y_test, 0, 1)
dtc_errors = np.where(y_pred_dtc == Y_test, 0, 1)
mlp_errors = np.where(y_pred_mlp == Y_test, 0, 1)
```

Ottenuti tali vettori, si procede identificando quanti e quali sono gli errori comuni tra essi, ottenendo così gli errori che il modello basato su ensemble learning commetterà sicuramente. Le funzioni e l'approccio utilizzati sono simili a quelli applicati precedentemente.

```
nb_dtc_errors = np.where(nb_errors == dtc_errors, nb_errors, 0)
nb_mlp_errors = np.where(nb_errors == mlp_errors, nb_errors, 0)
dtc_mlp_errors = np.where(dtc_errors == mlp_errors, dtc_errors, 0)
nb_dtc_mlp_errors = np.where(nb_dtc_errors == mlp_errors, mlp_errors, 0)
```

Il vettore "nb_dtc_mlp_errors" contiene gli errori commessi dai tre modelli (Naive bayes, Decision Tree e Multilayer Perceptron), attraverso l'intersezione dei vari vettori. Per ottenere una maggiore leggibilità, tali intersezioni sono state graficate attraverso un diagramma di Venn.



Dal grafico è facilmente intuibile la quantità di errori in comune tra i modelli. Tali imprecisioni però, verranno commesse anche dal "votingClassifier", in quanto sfrutta le caratteristiche dei tre classificatori. Inoltre, dobbiamo considerare che i modelli non sono perfetti, compiono degli errori, alcuni **riducibili**, altri **irriducibili**.
Il classificatore a votazione, oltre ai 94 errori comuni, potrebbe commetterne ulteriori in fase di predizione, ciò potrebbe essere causato da diversi fattori, tra i principali identificati dal team troviamo **la natura troppo diversa** dei tre modelli utilizzati. I classificatori utilizzano tecniche molto diverse per compiere scelte durante la predizione.
A seguito di tutte queste considerazioni, il modello che riesce a soddisfare in maniera ideale gli obiettivi del progetto è il Multilayer Perceptron.
Tale classificatore quindi, è stato implementato all'interno della **demo** per mostrare il funzionamento del progetto.

16 Implementazione

Ultimo step del progetto "Findrange" consiste nel definire quale dei 4 modelli creati verrà usato per la fase di funzionamento.
Per i motivi citati in presenza, abbiamo deciso poi di usare l'algoritmo avente precisione maggiore rispetto agli altri, ovvero **Multi Layer Perceptron**, infatti, ciò lo si può scaturire dal risultato delle metriche di valutazione di tutti i modelli.
Pertanto, prima di tutto, ci siamo salvati in file esterni sia l'oggetto **transformer** (usato nella fase di **normalizzazione** dei dati) che l'algoritmo **mlp** (rappresenta il modello **Multi Layer Perceptron**), in modo da non dover, ogni volta, rieseguire l'intero file riguardante la preparazione e l'addestramento del modello.
Tutto ciò, è stato fatto con l'aiuto della libreria **joblib**, la quale offre il metodo **dump()**, per compiere il salvataggio in un file esterno con estensione .joblib.
I file joblib sono un formato di file utilizzati per salvare e caricare oggetti Python in maniera efficiente; sono ottimizzati per la grande quantità di dati che possono archiviare e offrono, tramite la compressione, una riduzione delle dimensioni del file.
Di seguito mostrato il funzionamento.

```
dump(transformer, "transformer.joblib")
```

```
dump(mlp, 'price_analizer.joblib')
```

Dopodichè, abbiamo proceduto ad implementare la Demo, ovvero la parte in cui l'utente, che userà il modello, dovrà fornire in input le variabili indipendenti, per poi consultare l'output stampato a video rendendosi conto della fascia di prezzo che andrà ad affrontare poi nella realtà.
Per fare ciò abbiamo usato un file python, denominato **implementator.py**.
Di seguito mostrato il codice.

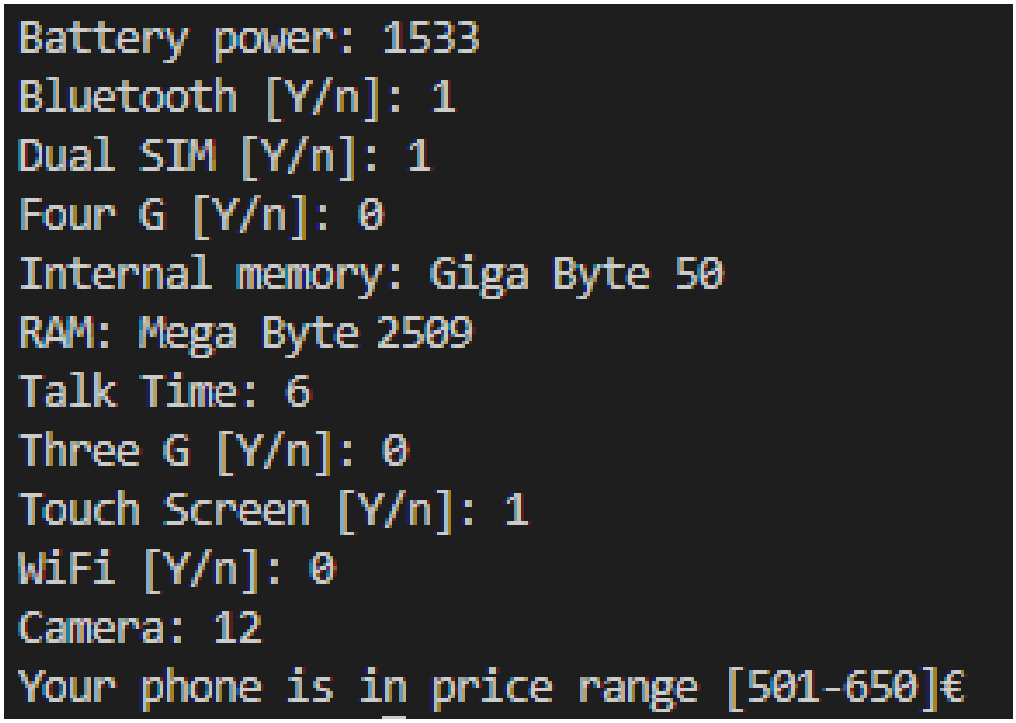
```python
mlp: MLPClassifier = load('price_analizer.joblib')
transformer: MinMaxScaler = load("transformer.joblib")
while True:
    battery_power = float(input("Battery power: "))
    blue = float(input("Bluetooth [Y/n]: "))
    dual_sim = float(input("Dual SIM [Y/n]: "))
    four_g = float(input("Four G [Y/n]: "))
    int_memory = float(input("Internal memory: Giga Byte "))
    ram = float(input("RAM: Mega Byte"))
    talk_time = float(input("Talk Time: "))
    three_g = float(input("Three G [Y/n]: "))
    touch_screen = float(input("Touch Screen [Y/n]: "))
    wifi = float(input("WiFi [Y/n]: "))
    camera = float(input("Camera: "))
    phone = pd.DataFrame({"battery_power": [battery_power], "blue": [blue],
    "dual_sim": [dual_sim], "four_g": [four_g], "int_memory": [int_memory], "ram": [ram],
    "talk_time": [talk_time], "three_g": [three_g], "touch_screen": [touch_screen],
    "wifi": [wifi], "camera": [camera]})

    normalized_phone = transformer.transform(phone)
    prediction = mlp.predict(normalized_phone)

    if prediction[0]==0:
        print("Your phone is in price range [200-350]€")
    elif prediction[0]==1:
        print("Your phone is in price range [351-500]€")
    elif prediction[0]==2:
        print("Your phone is in price range [501-650]€")
    else:
        print("Your phone is in price range [650-800]€")
```

Innanzitutto ci carichiamo gli oggetti Python che abbiamo salvato con il metodo dump(), tramite il metodo **load()**, il quale è anch'esso un metodo offerto dalla libreria joblib.
Iniziamo con un **while true**, un ciclo all'infinito, nel quale l'utente ci dovrà fornire in input tutte le variabili indipendenti. Per le variabili [Y/n] ci dovrà fornire solo uno tra i valori 1,0 che significano rispettivamente Yes e No, mentre per le altre variabili, un valore numerico.
Ci siamo creati, poi, un DataFrame della libreria pandas chiamato **phone**, assegnando i valori per ogni colonna. Rappresenta, in pratica, un oggetto contenente tutti i valori forniti in input dall'utente.
Attenendoci a questi dati, però, ci siamo accorti che l'output del modello era sempre 3, ovvero il range di prezzo massimo previsto dal modello, anche se l'istanza del Dataset non confermava ciò. Ci siamo accorti, infatti, che i dati, inseriti in input, non erano normalizzati, pertanto capitava che se l'utente avesse inserito 700 come "battery_power", ma nel Dataset tale valore era normalizzato nell'intervallo [0-1], il modello riconosceva questo attributo "altissimo" e quindi tendeva a fornire, come output, il range di prezzo più alto.
Per risolvere il problema, abbiamo normalizzato anche i dati forniti dall'utente,

prima di confrontarli con le varie istanze, in modo che siano conformi a quelli presenti nel Dataset (anch'esso normalizzato).
Abbiamo poi invocato il metodo **predict()** su **mlp**, che rappresenta il modello **Multi Layer Perceptron**, per farci restituire la variabile dipendente predetta. Infine, abbiamo effettuato una stampa diversa per ogni output della variabile dipendente, allo scopo di mostrare a video direttamente il range di prezzo prestabilito nella fase di Documentazione Dati.

16.1 Screen del funzionamento

Inseriamo tutti i valori delle variabili indipendenti richiesti in input e l'output sarà il range nel quale verterà il prezzo del dispositivo avente tali caratteristiche. Di seguito mostrato il funzionamento.



17 Link Github

Di seguito riportiamo il link al nostro repository Github.
`https://github.com/AngeloSantangelo/Findrange`